Notações: in, pré e posfixada

Com base no que vimos, construa um programa, na linguagem C, que leia da entrada padrão uma string representando uma expressão infixada, com presença de parênteses, a converta em posfixada e a avalie retornando na saída padrão o resultado da avaliação.

```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#define MAXCOLS 80
typedef struct nodo
{
    int inf;
    struct nodo * next;
}NODO;
typedef NODO * PILHA_ENC;
void cria_pilha (PILHA_ENC *);
int eh_vazia (PILHA_ENC);
void push (PILHA_ENC *, int);
int top (PILHA_ENC);
void pop (PILHA_ENC *);
int top_pop (PILHA_ENC *);
void destruir (PILHA_ENC);
```

```c
int avaliar (char *);
int eh_operando(char);
int aplicar (int, char, int);
int prcd (char, char);
void converter(char *, char *);
main()  {
    char expr [MAXCOLS], expr2[MAXCOLS];
    int position = 0;
    while ((expr[position++] = getchar ()) != '\n');
    expr[--position]='\0';
    printf ("%s%s","a expressão infixada original eh ",expr);
    converter(expr,expr2);
    printf ("\n%s%s","a expressão posfixada eh ",expr2);
    printf (" = %d\n",avaliar(expr2));
}
```

```c
void cria_pilha (PILHA_ENC *pp) {
    *pp=NULL;
}
int eh_vazia (PILHA_ENC p) {
    return (!p);
}
void push (PILHA_ENC *pp, int v)
{
    NODO *novo;
    novo = (NODO *) malloc (sizeof(NODO));
    if (!novo)     {
        printf ("\nERRO! Memoria insuficiente!\n");
        exit (1);
    }
    novo->inf = v;
    novo->next = *pp;
    *pp=novo;
}
```

```c
int top (PILHA_ENC p) {
    if (eh_vazia(p)) {
        printf ("\nERRO! Consulta em pilha vazia!\n");
        exit (2);
    } else
        return (p->inf);
}
void pop (PILHA_ENC *pp) {
    if (eh_vazia(*pp)) {
        printf ("\nERRO! Retirada em pilha vazia!\n");
        exit (3);
    } else {
        NODO *aux=*pp;
        *pp=(*pp)->next;
        free (aux);
    }
}
```

```c
int top_pop (PILHA_ENC *pp)
{
    if (eh_vazia(*pp))
    {
        printf ("\nERRO! Consulta e retirada em pilha vazia!\n");
        exit (4);
    }
    else
    {
        int v=(*pp)->inf;
        NODO *aux=*pp;
        *pp=(*pp)->next;
        free (aux);
        return (v);
    }
}
```

```c
void destruir (PILHA_ENC l)
{
    PILHA_ENC aux;
    while (l)
    {
        aux = l;
        l = l->next;
        free(aux);
    }
}
```

```
1  int eh_operando(char op)
2  {
3      return (op != '+' && op != '-' &&
4      op != '*' && op != '/' && op != '^' &&
5      op != '(' && op != ')');
6  }
```

```
1  int aplicar (int operando1, char operador,
2  int operando2) {
3      switch (operador)
4      {
5          case '+': return (operando1 + operando2);
6          case '-': return (operando1 - operando2);
7          case '*': return (operando1 * operando2);
8          case '/': return (operando1 / operando2);
9          case '^': return ((int)pow(operando1, operando2));
10     }
11 }
```

```c
int avaliar (char *e) {
    char symbol;
    int i=0;
    PILHA_ENC pilha_operandos;
    cria_pilha (&pilha_operandos);
    while (symbol = e[i++])
        if (eh_operando(symbol))
            push (&pilha_operandos, symbol-'0');
        else {
            int op2=top_pop(&pilha_operandos),
            op1=top_pop(&pilha_operandos);
            push (&pilha_operandos, aplicar (op1, symbol, op2));
        }
    return (top_pop(&pilha_operandos));
}
```

```
int prcd (char op1, char op2)
{
    if ((op1=='+' || op1=='-') && (op2=='+' ||
    op2=='-'))
        return (1);
    if ((op1=='*' || op1=='/') && (op2=='+' ||
    op2=='-' || op2=='*' || op2=='/'))
        return (1);
    if (op1=='^' && (op2=='^' || op2=='+' ||
    op2=='-' || op2=='*' || op2=='/'))
        return (1);
    if ((op1=='+' || op1=='-' || op1=='*' || op1=='/' ||
    op1=='^') && op2==')')
        return (1);
    return (0);
}
```

```
1  void converter(char *o, char *d) {
2      char symbol;
3      int i1=0, i2=0;
4      PILHA_ENC opstk;
5      cria_pilha (&opstk);
6      while (symbol=o[i1++])
7          if (eh_operando(symbol))
8              d[i2++]=symbol;
9          else {
10             while (!eh_vazia(opstk) && prcd (top(opstk),symbol))
11                 d[i2++]=(char)top_pop(&opstk);
12             if (symbol==')')
13                 pop(&opstk);
14             else
15                 push(&opstk, symbol);
16         }
17     while (!eh_vazia(opstk))
18         d[i2++]=(char)top_pop(&opstk);
19     d[i2]='\0';
20 }
```

Árvores – Caracterização
Árvores Binárias

Árvores - Conceitos

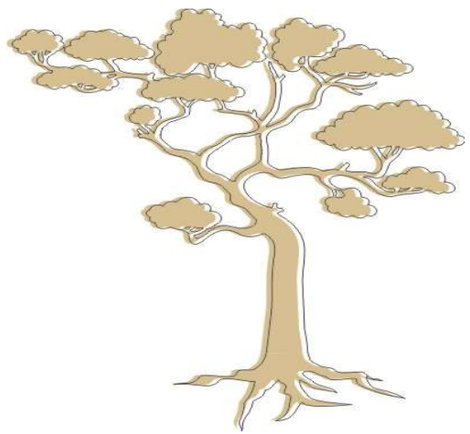
Qual a carência das pilhas e filas?

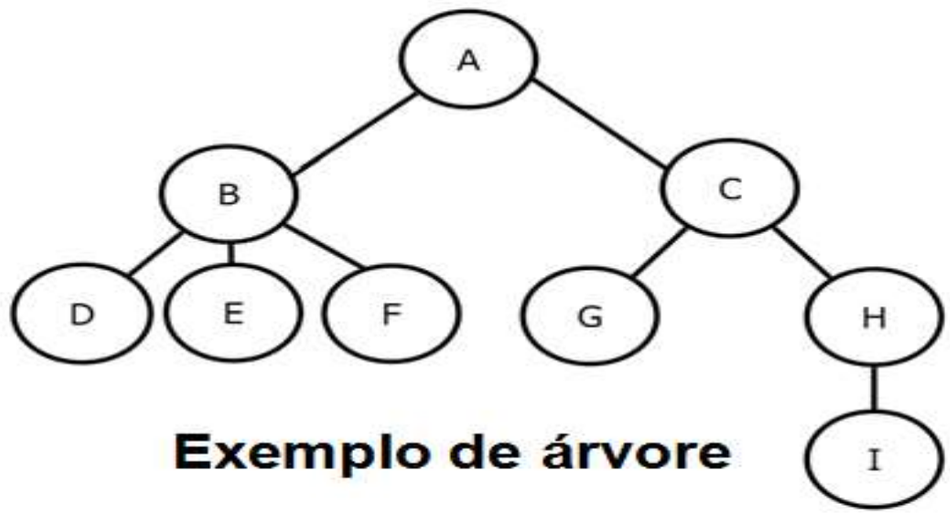
Estas são de difícil utilização para a representação hierárquica de elementos.

Devido a...

Serem limitadas a apenas uma dimensão.

Visando eliminar esta limitação foi criado o conceito de árvore.

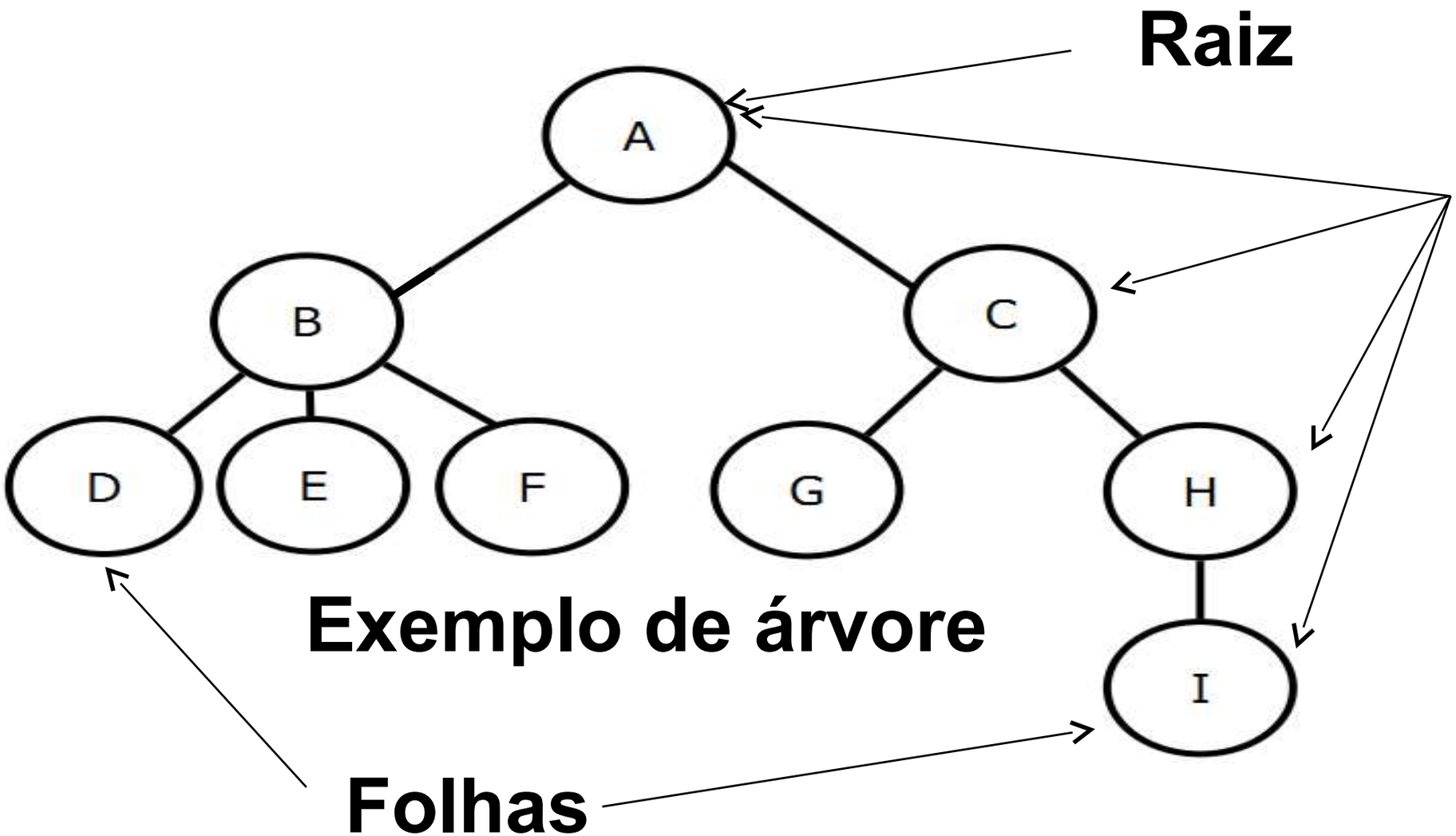




Exemplo de árvore

Árvores - Conceitos

Conceitos:



Exemplo de árvore

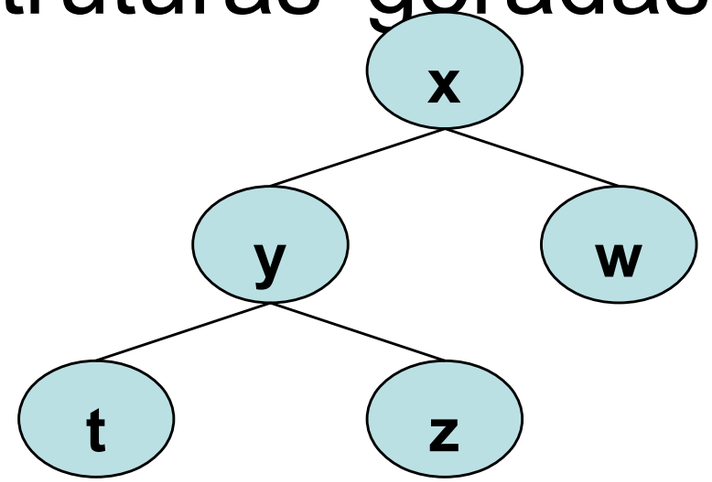
Árvores - Conceitos

Uma definição recursiva para árvore é:

1. Uma estrutura vazia é uma árvore.

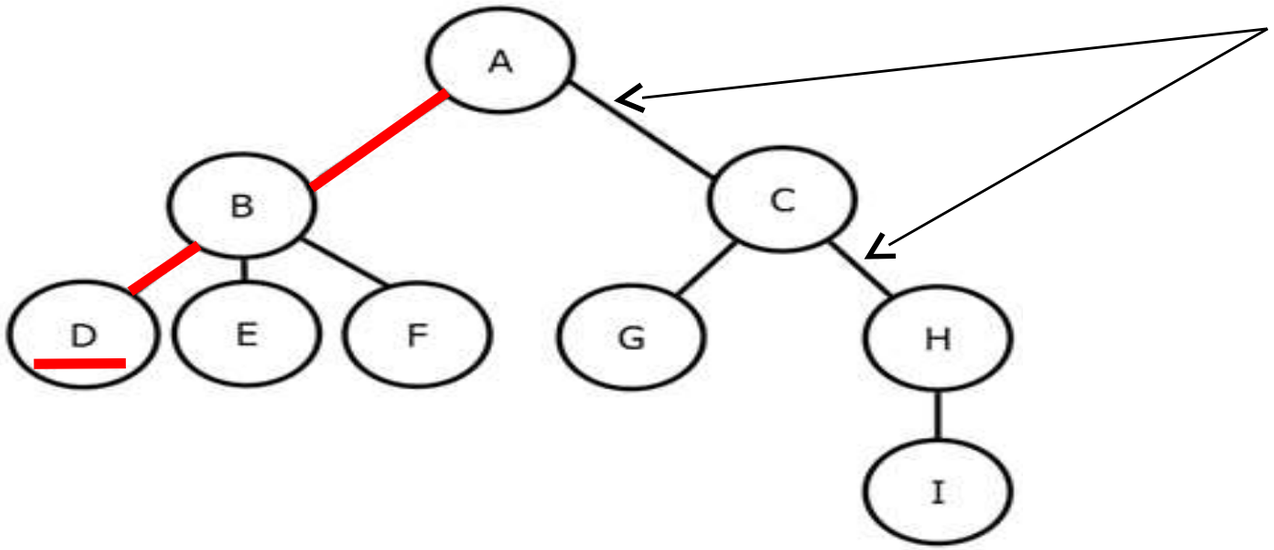
2. Se t_1, t_2, …, t_k são árvores disjuntas, então a estrutura cuja raiz tem como suas filhas as raízes de t_1, t_2, …, t_k também é uma árvore.

3. Somente estruturas geradas pelas regras 1 e 2 são árvores.



Árvores - Conceitos

Qual o conceito de caminho?



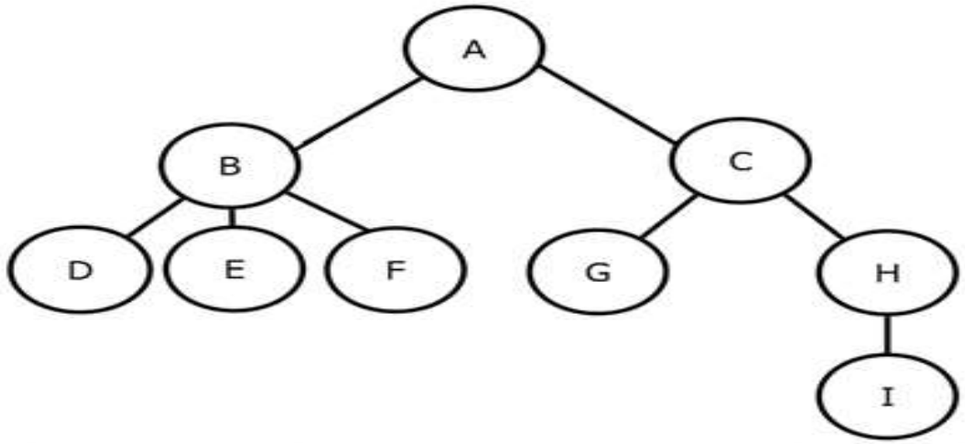
arcos

Caminho é a sequência de arcos, com origem na raiz e final em um determinado nó.

Quantos caminhos existem para se atingir um determinado nó?

Apenas um.

Árvores - Conceitos



O que determina o tamanho de um caminho?

O número de arcos no mesmo.

Qual o nível de um determinado nó?

O nível de um nó em uma árvore é definido da seguinte forma: a raiz da árvore tem nível zero e o nível de qualquer outro nó na árvore é um nível a mais que o de seu pai.

Árvores - Conceitos

Qual a altura (ou profundidade) de uma árvore não vazia?

O nível máximo de um nó na árvore.

A árvore vazia é uma árvore legítima de altura 0 (zero).

A definição de árvore impõe alguma restrição sobre a quantidade de filhos de um nó?

Não. Esta pode variar de zero até qualquer inteiro positivo.

Porém, existem alguns tipos particulares de árvores, cuja suas definições impõem algumas restrições. Por exemplo, árvores binárias.

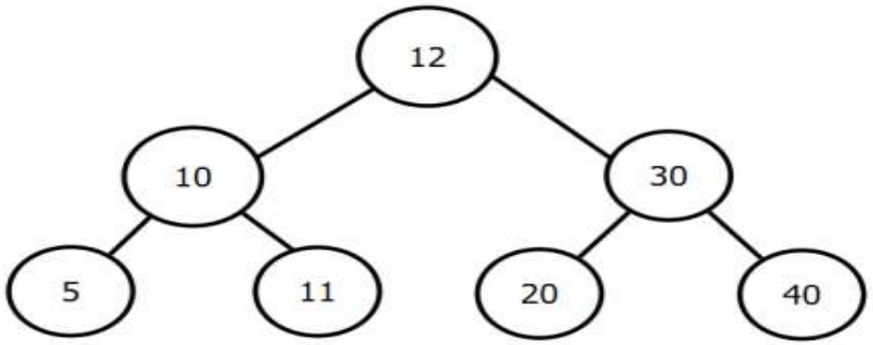
UNIVASF

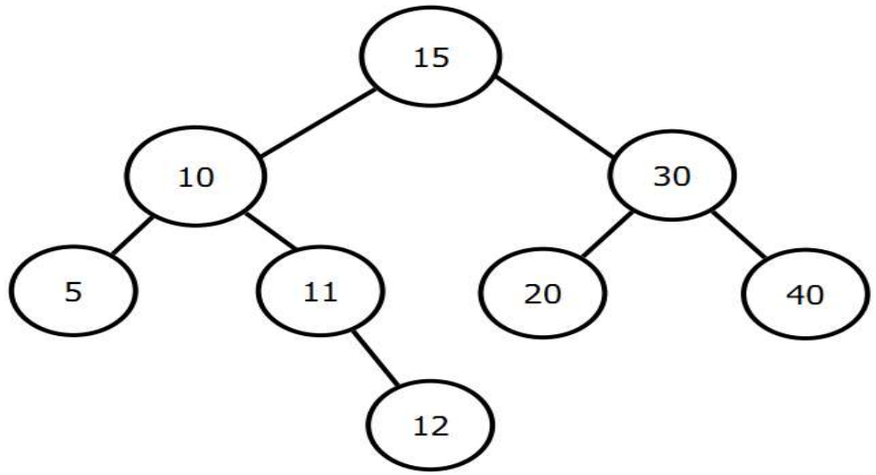
Árvores Binárias

O que é uma árvore binária?

É uma árvore cujos nós têm dois filhos (possivelmente vazios) e cada filho é designado como filho à esquerda ou filho à direita.

Ex.:





Em uma árvore binária existem no máximo quantos nós em um determinado nível?

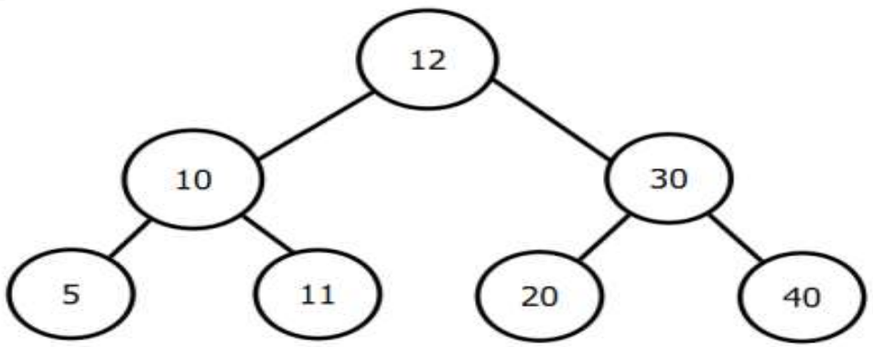
Existem no máximo 2^i nós no nível *i*.

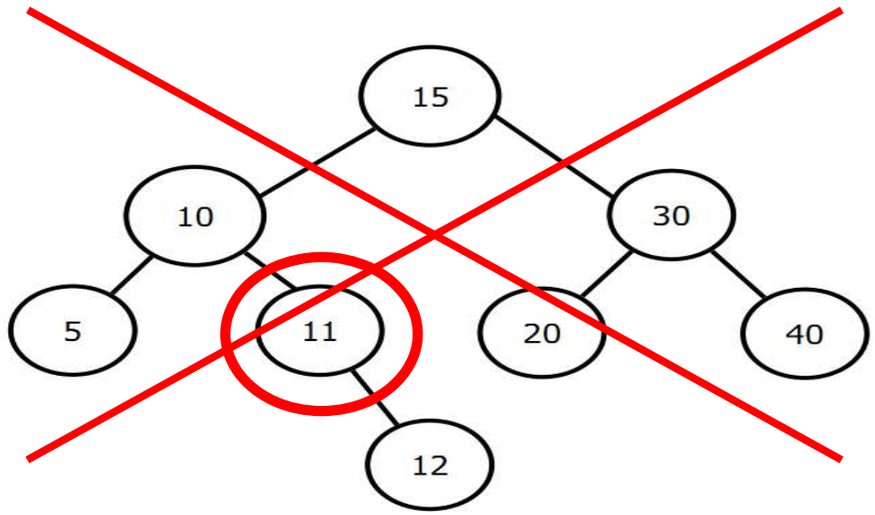
Árvores Binárias

Árvore estritamente binária

É uma árvore onde todos os nós que não são folha possuem dois filhos.

Ex.





Árvore binária completa

Uma árvore binária completa de profundidade *d* é uma árvore estritamente binária onde todas as folhas estão no nível *d*.

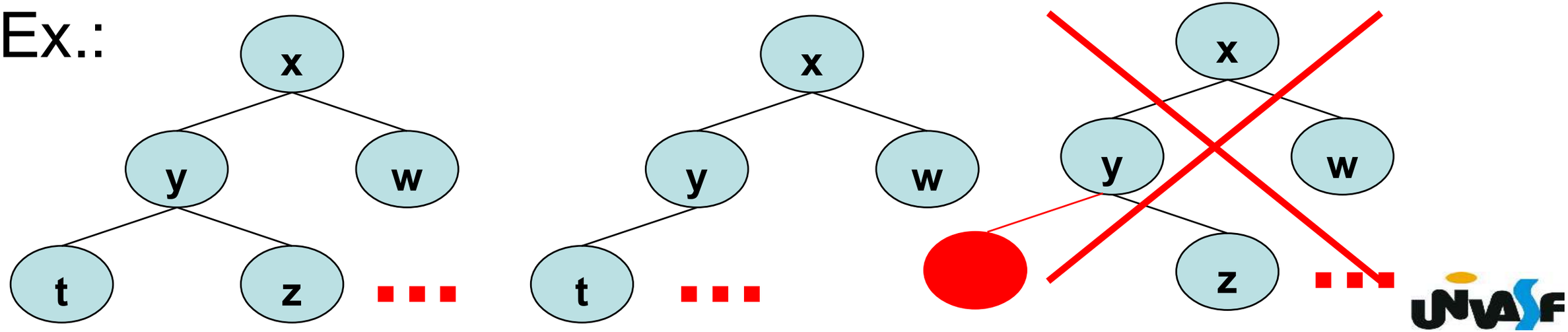
Árvores Binárias

Árvore binária quase completa

Uma árvore binária de profundidade *d* será uma árvore binária quase completa se:

1. **Cada folha da árvore estiver no nível *d* ou no nível *d*-1.**
2. **Para todo nó *nd* que possui um descendente direito no nível *d*, todo descendente esquerdo de *nd* é folha no nível *d* ou tem 2 filhos.**

Ex.:



Árvores Binárias
Armazenamento Estático

Árvores Binárias

Assim como vimos em nosso estudo sobre listas, árvores também podem ser armazenadas de forma estática ou dinâmica.

Começaremos nosso estudo com o armazenamento estático.

Quais campos seriam relevantes para cada nó?

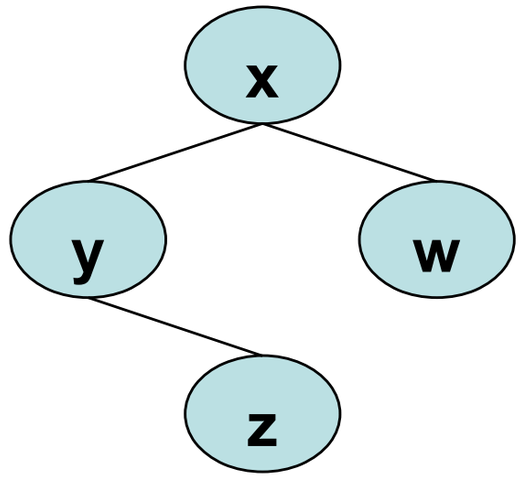
Os campos *info*, *left*, *right* e *father*.

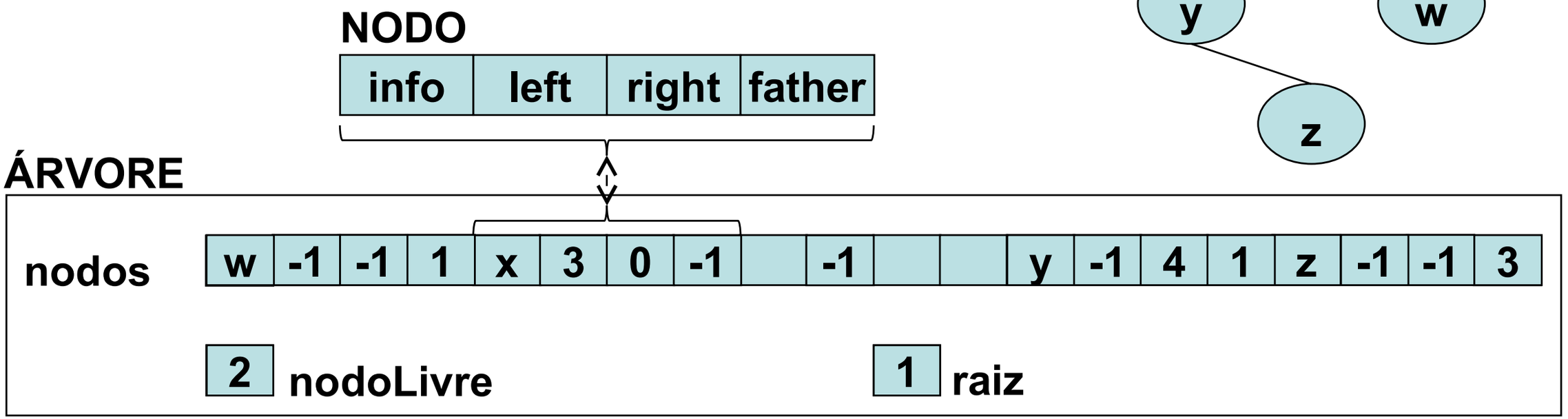
Árvores Binárias

Como vocês sugeririam o armazenamento estático?

Em um vetor?

Como?





Árvores Binárias

Quais operações primitivas seriam relevantes?

Se ***p*** é uma referência para um nó ***nd*** de uma árvore binária associada a ***t***, temos:

- a função ***info(t, p)*** retorna o conteúdo de ***nd***.

- as funções ***left(t, p)***, ***right(t, p)***, ***father(t, p)*** e ***brother(t, p)***. Estas funções retornaram referência inválidas se ***nd*** não tiver filho esquerdo, filho direito, pai ou irmão, respectivamente.

Árvores Binárias

Temos também as funções lógicas

isleft(t, p) e ***isright(t, p)***.

Para construir uma árvore binária, as operações ***maketree(t, x)***, ***setleft(t, p, x)*** e ***setright(t, p, x)*** são úteis.

Com o que foi definido podemos implementar, na linguagem C, uma árvore binária como sendo:

```c
typedef struct {
    int info;
    int left;
    int right;
    int father;
} NODE;
typedef struct{
    int root;
    int nodeFree;
    NODE nodes[NUMNODES];/*#define NUMNODES 100*/
}ARV_BIN_SEQ;
void maketree(ARV_BIN_SEQ *, int);
void setleft(ARV_BIN_SEQ *, int, int);
void setright(ARV_BIN_SEQ *, int, int);
int info(ARV_BIN_SEQ *, int);
int left(ARV_BIN_SEQ *, int);
int right(ARV_BIN_SEQ *, int);
int father(ARV_BIN_SEQ *, int);
int brother(ARV_BIN_SEQ *, int);
int isleft(ARV_BIN_SEQ *, int);
int isright(ARV_BIN_SEQ *, int);
```

Árvores Binárias - Alocação Sequencial

Com base no que foi visto implemente as operações que compõem o TAD ARV_BIN_SEQ.

```
1   typedef struct {
2       int info;
3       int left;
4       int right;
5       int father;
6   } NODE;
7   typedef struct{
8       int root;
9       int nodeFree;
10      NODE nodes[NUMNODES];/*#define NUMNODES 100*/
11  }ARV_BIN_SEQ;
12  void maketree(ARV_BIN_SEQ *, int);
13  void setleft(ARV_BIN_SEQ *, int, int);
14  void setright(ARV_BIN_SEQ *, int, int);
15  int info(ARV_BIN_SEQ *, int);
16  int left(ARV_BIN_SEQ *, int);
17  int right(ARV_BIN_SEQ *, int);
18  int father(ARV_BIN_SEQ *, int);
19  int brother(ARV_BIN_SEQ *, int);
20  int isleft(ARV_BIN_SEQ *, int);
21  int isright(ARV_BIN_SEQ *, int);
```

```c
void maketree(ARV_BIN_SEQ *t, int x)
{
    int i, ind;
    for (i=0; i<NUMNODES-1; i++)
        t->nodes[i].left = i+1;
    t->nodes[i].left = -1;
    t->nodeFree=0;
    ind = getNode(t);
    if (ind != -1) {
        t->nodes[ind].info = x;
        t->nodes[ind].left = -1;
        t->nodes[ind].right = -1;
        t->nodes[ind].father = -1;
        t->root = ind;
    } else {
        printf("Impossivel construir a arvore!");
        exit(1);
    }
}
```

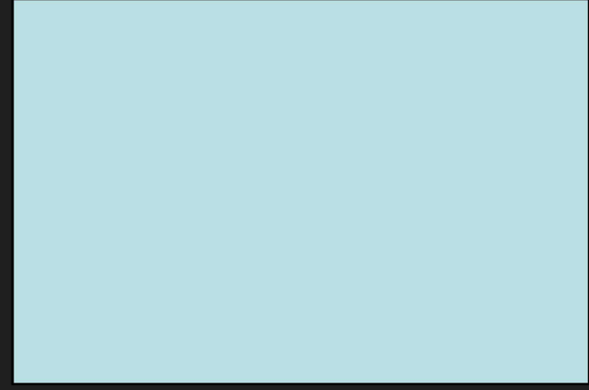
```
1   int getNode(ARV_BIN_SEQ *t)
2   {
3       if (t->nodeFree != -1)
4       {
5           int i = t->nodeFree;
6           t->nodeFree = t->nodes[t->nodeFree].left;
7           return i;
8       }
9       else
10          return -1;
11  }
12  void freeNode(ARV_BIN_SEQ *t, int node)
```

Árvores Binárias - Alocação Sequencial

Com base no que foi visto implemente as operações que compõem o TAD ARV_BIN_SEQ.

```
1   typedef struct {
2       int info;
3       int left;
4       int right;
5       int father;
6   } NODE;
7   typedef struct{
8       int root;
9       int nodeFree;
10      NODE nodes[NUMNODES];/*#define NUMNODES 100*/
11  }ARV_BIN_SEQ;
12  void maketree(ARV_BIN_SEQ *, int);
13  void setleft(ARV_BIN_SEQ *, int, int);
14  void setright(ARV_BIN_SEQ *, int, int);
15  int info(ARV_BIN_SEQ *, int);
16  int left(ARV_BIN_SEQ *, int);
17  int right(ARV_BIN_SEQ *, int);
18  int father(ARV_BIN_SEQ *, int);
19  int brother(ARV_BIN_SEQ *, int);
20  int isleft(ARV_BIN_SEQ *, int);
21  int isright(ARV_BIN_SEQ *, int);
```

```c
void setleft(ARV_BIN_SEQ *t, int p, int x)
{
    int ind = getNode(t);
    if (ind != -1) {
        t->nodes[p].left = ind;
        t->nodes[ind].info = x;
        t->nodes[ind].left = -1;
        t->nodes[ind].right = -1;
        t->nodes[ind].father = p;
    } else {
        printf("Impossivel inserir filho a esquerda!");
        exit(2);
    }
}
```



Árvores Binárias - Alocação Sequencial

Com base no que foi visto implemente as operações que compõem o TAD ARV_BIN_SEQ.

```
1   typedef struct {
2       int info;
3       int left;
4       int right;
5       int father;
6   } NODE;
7   typedef struct{
8       int root;
9       int nodeFree;
10      NODE nodes[NUMNODES];/*#define NUMNODES 100*/
11  }ARV_BIN_SEQ;
12  void maketree(ARV_BIN_SEQ *, int);
13  void setleft(ARV_BIN_SEQ *, int, int);
14  void setright(ARV_BIN_SEQ *, int, int);
15  int info(ARV_BIN_SEQ *, int);
16  int left(ARV_BIN_SEQ *, int);
17  int right(ARV_BIN_SEQ *, int);
18  int father(ARV_BIN_SEQ *, int);
19  int brother(ARV_BIN_SEQ *, int);
20  int isleft(ARV_BIN_SEQ *, int);
21  int isright(ARV_BIN_SEQ *, int);
```

```c
void setright(ARV_BIN_SEQ *t, int p, int x)
{
    int ind = getNode(t);
    if (ind != -1) {
        t->nodes[p].right = ind;
        t->nodes[ind].info = x;
        t->nodes[ind].left = -1;
        t->nodes[ind].right = -1;
        t->nodes[ind].father = p;
    } else {
        printf("Impossivel inserir filho a direita!");
        exit(2);
    }
}
```

Árvores Binárias - Alocação Sequencial

Com base no que foi visto implemente as operações que compõem o TAD ARV_BIN_SEQ.

```
1   typedef struct {
2       int info;
3       int left;
4       int right;
5       int father;
6   } NODE;
7   typedef struct{
8       int root;
9       int nodeFree;
10      NODE nodes[NUMNODES];/*#define NUMNODES 100*/
11  }ARV_BIN_SEQ;
12  void maketree(ARV_BIN_SEQ *, int);
13  void setleft(ARV_BIN_SEQ *, int, int);
14  void setright(ARV_BIN_SEQ *, int, int);
15  int info(ARV_BIN_SEQ *, int);
16  int left(ARV_BIN_SEQ *, int);
17  int right(ARV_BIN_SEQ *, int);
18  int father(ARV_BIN_SEQ *, int);
19  int brother(ARV_BIN_SEQ *, int);
20  int isleft(ARV_BIN_SEQ *, int);
21  int isright(ARV_BIN_SEQ *, int);
```

```c
int info(ARV_BIN_SEQ *t, int p)
{
    return t->nodes[p].info;
}
```

Árvores Binárias - Alocação Sequencial

Com base no que foi visto implemente as operações que compõem o TAD ARV_BIN_SEQ.

```
1   typedef struct {
2       int info;
3       int left;
4       int right;
5       int father;
6   } NODE;
7   typedef struct{
8       int root;
9       int nodeFree;
10      NODE nodes[NUMNODES];/*#define NUMNODES 100*/
11  }ARV_BIN_SEQ;
12  void maketree(ARV_BIN_SEQ *, int);
13  void setleft(ARV_BIN_SEQ *, int, int);
14  void setright(ARV_BIN_SEQ *, int, int);
15  int info(ARV_BIN_SEQ *, int);
16  int left(ARV_BIN_SEQ *, int);
17  int right(ARV_BIN_SEQ *, int);
18  int father(ARV_BIN_SEQ *, int);
19  int brother(ARV_BIN_SEQ *, int);
20  int isleft(ARV_BIN_SEQ *, int);
21  int isright(ARV_BIN_SEQ *, int);
```

```c
int left(ARV_BIN_SEQ *t, int p)
{
    return t->nodes[p].left;
}
```

Árvores Binárias - Alocação Sequencial

Com base no que foi visto implemente as operações que compõem o TAD ARV_BIN_SEQ.

```
1   typedef struct {
2       int info;
3       int left;
4       int right;
5       int father;
6   } NODE;
7   typedef struct{
8       int root;
9       int nodeFree;
10      NODE nodes[NUMNODES];/*#define NUMNODES 100*/
11  }ARV_BIN_SEQ;
12  void maketree(ARV_BIN_SEQ *, int);
13  void setleft(ARV_BIN_SEQ *, int, int);
14  void setright(ARV_BIN_SEQ *, int, int);
15  int info(ARV_BIN_SEQ *, int);
16  int left(ARV_BIN_SEQ *, int);
17  int right(ARV_BIN_SEQ *, int);
18  int father(ARV_BIN_SEQ *, int);
19  int brother(ARV_BIN_SEQ *, int);
20  int isleft(ARV_BIN_SEQ *, int);
21  int isright(ARV_BIN_SEQ *, int);
```

```c
int right(ARV_BIN_SEQ *t, int p)
{
    return t->nodes[p].right;
}
```

Árvores Binárias - Alocação Sequencial

Com base no que foi visto implemente as operações que compõem o TAD ARV_BIN_SEQ.

```
1   typedef struct {
2       int info;
3       int left;
4       int right;
5       int father;
6   } NODE;
7   typedef struct{
8       int root;
9       int nodeFree;
10      NODE nodes[NUMNODES];/*#define NUMNODES 100*/
11  }ARV_BIN_SEQ;
12  void maketree(ARV_BIN_SEQ *, int);
13  void setleft(ARV_BIN_SEQ *, int, int);
14  void setright(ARV_BIN_SEQ *, int, int);
15  int info(ARV_BIN_SEQ *, int);
16  int left(ARV_BIN_SEQ *, int);
17  int right(ARV_BIN_SEQ *, int);
18  int father(ARV_BIN_SEQ *, int);
19  int brother(ARV_BIN_SEQ *, int);
20  int isleft(ARV_BIN_SEQ *, int);
21  int isright(ARV_BIN_SEQ *, int);
```

```c
int father(ARV_BIN_SEQ *t, int p)
{
    return t->nodes[p].father;
}
```

Árvores Binárias - Alocação Sequencial

Com base no que foi visto implemente as operações que compõem o TAD ARV_BIN_SEQ.

```
1   typedef struct {
2       int info;
3       int left;
4       int right;
5       int father;
6   } NODE;
7   typedef struct{
8       int root;
9       int nodeFree;
10      NODE nodes[NUMNODES];/*#define NUMNODES 100*/
11  }ARV_BIN_SEQ;
12  void maketree(ARV_BIN_SEQ *, int);
13  void setleft(ARV_BIN_SEQ *, int, int);
14  void setright(ARV_BIN_SEQ *, int, int);
15  int info(ARV_BIN_SEQ *, int);
16  int left(ARV_BIN_SEQ *, int);
17  int right(ARV_BIN_SEQ *, int);
18  int father(ARV_BIN_SEQ *, int);
19  int brother(ARV_BIN_SEQ *, int);
20  int isleft(ARV_BIN_SEQ *, int);
21  int isright(ARV_BIN_SEQ *, int);
```

Árvores Binárias - Alocação Sequencial

Com base no que foi visto implemente as operações que compõem o TAD ARV_BIN_SEQ.

```
1   typedef struct {
2       int info;
3       int left;
4       int right;
5       int father;
6   } NODE;
7   typedef struct{
8       int root;
9       int nodeFree;
10      NODE nodes[NUMNODES];/*#define NUMNODES 100*/
11  }ARV_BIN_SEQ;
12  void maketree(ARV_BIN_SEQ *, int);
13  void setleft(ARV_BIN_SEQ *, int, int);
14  void setright(ARV_BIN_SEQ *, int, int);
15  int info(ARV_BIN_SEQ *, int);
16  int left(ARV_BIN_SEQ *, int);
17  int right(ARV_BIN_SEQ *, int);
18  int father(ARV_BIN_SEQ *, int);
19  int brother(ARV_BIN_SEQ *, int);
20  int isleft(ARV_BIN_SEQ *, int);
21  int isright(ARV_BIN_SEQ *, int);
```

Árvores Binárias - Alocação Sequencial

Com base no que foi visto implemente as operações que compõem o TAD ARV_BIN_SEQ.

```
1   typedef struct {
2       int info;
3       int left;
4       int right;
5       int father;
6   } NODE;
7   typedef struct{
8       int root;
9       int nodeFree;
10      NODE nodes[NUMNODES];/*#define NUMNODES 100*/
11  }ARV_BIN_SEQ;
12  void maketree(ARV_BIN_SEQ *, int);
13  void setleft(ARV_BIN_SEQ *, int, int);
14  void setright(ARV_BIN_SEQ *, int, int);
15  int info(ARV_BIN_SEQ *, int);
16  int left(ARV_BIN_SEQ *, int);
17  int right(ARV_BIN_SEQ *, int);
18  int father(ARV_BIN_SEQ *, int);
19  int brother(ARV_BIN_SEQ *, int);
20  int isleft(ARV_BIN_SEQ *, int);
21  int isright(ARV_BIN_SEQ *, int);
```